The Composition Pattern for model-driven robot application development

Dominick Vanthienen and Herman Bruyninckx

KU Leuven, Dept. of Mechanical Engineering, PMA Division
Celestijnenlaan 300B, 3001 Leuven, Belgium
dominick.vanthienen@kuleuven.be **

Robots evolve to more autonomous systems that can operate in human-populated environments. Moreover, these new environments and related applications expect physical and cognitive interaction capabilities of the robot. This evolution requires the integration of many different fields of research to be carried out by a multitude of experts. In order to bring this integration challenge to a good end, a systematic and model-driven approach to software is needed, which should result in flexible, reusable, and adaptable software. This paper describes the Composition Pattern [3] as such a systematic approach to model robot applications. It is an architectural pattern to systematically structure, i.e. to contain and connect, types of behavior. It builds on and provides a constructive way to apply the 5C's approach to separation of concerns [2].

The pattern can be applied to analyse, model and implement applications. Earlier work introduced the Composition Pattern as software architectural pattern [4]. The focus of this paper and presentation will be on the modeling aspect.

1 The Composition Pattern

The Composition Pattern builds on four concepts, i.e. metamodeling, composition, hierarchy, and semantic context, explained in following paragraphs.

The first concept is **metamodeling** [1], an approach from Model Driven Engineering (MDE) which separates domain knowledge from its software implementation, and formalizes this knowledge in a meta-model. The meta-model forms a Domain Specific (modeling) Language (DSL) to describe specific models. The conformance of a model to one or more meta-models can be verified. From a model, an implementation in the software framework of choice can be hand-coded or generated.

The second concept is **composition**. The Composition Pattern defines a structure to compose *entities* (i.e. *models* in the context of this paper). Each of these entities is of one of the following types (of behavior).

- *Functional Entities* model continuous time and space behavior, i.e. 'data processing' or 'computations'. A Functional Entity can be a *composite (Func-*

** All authors gratefully acknowledge the financial support by the Flemish FWO project G040410N, KU Leuven's Concerted Research Action GOA/2010/011, and European FP7 project Factory-in-a-Day (grant agreement no. 609206).

tional Entity), composing other Functional Entities and 'support entities'. The latter consist of entities of the types listed below.

- *Monitors* model conditions to verify on data and the events to raise based on these conditions.
- A *Coordinator* models the actions to command from the other entities within a composite. It gives the composite the autonomy to handle certain situations locally. It is a pure 'event processor', it receives events from the other entities and triggers (commands) other entities with events.
- A *Scheduler* models the resource access and timing constraints on the different entities within a composite.
- *Configurators* model different sets of settings, i.e. data and parameters to apply to an entity when triggered by the Coordinator. In that sense it 'translates' the event to the parameters to apply. A Configurator forms the point where knowledge from a knowledge base can be introduced.
- A *Composer* models how the entities within a composite are connected.
- *Communicators* model constraints on how entities exchange data and events over these connections. Data and event communication is not limited to the boundaries of a composite.

Figures 1 and 2 detail how these entity types (behavior) is structured by the Composition Pattern.

The third concept is **hierarchy**. Since each Functional Entity can be (replaced by) a composite Functional Entity, a tree of entities emerges with a recurring structure. The tree depth level does not have to be identical for all branched of the tree. The composition is hierarchical, however data and event communication is not hindered by the hierarchy: events are broadcast and data is communicated through the boundary of a composite.

The fourth concept is **semantic context**. The entities within a composite need to use a shared vocabulary, i.e. its semantic context, to be able to interact. Making this context explicit is important to apply knowledge driven approaches. The support entities handle the translation from the context of a composite to its child functional entities. The composite and its semantic context forms a boundary to what the support entities have to 'know' and manage. However, this boundary does not imply information hiding; child entities can be introspected and hence reasoned about.

2 Use cases and discussion

In following example we model the 'reaching task' part of a robotic pick and place application using the Composition Pattern. This reaching task is a *Functional Entity* which is part of an application composite. It is in itself a composite Functional Entity, composing a controller (composite) Functional Entity, a trajectory planner (composite) Functional Entity, and 'support entities'. For example following support entities and example interactions of their implementations: A *Monitor* monitors the control error and signals when it reaches a certain limit. The *Coordinator* reacts on this event and sends out an event to adapt the gains

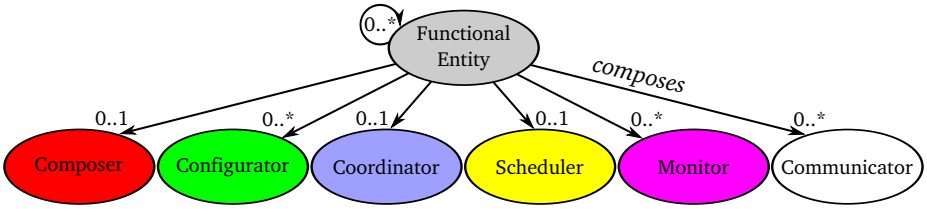

Fig. 1: Entity types within the Composition Pattern. A Functional Entity can compose a number of entities of each type, indicated by the arrow and the cardinality. A Functional Entity can be a composite Functional Entity, composing other Functional Entities, as indicated by the reflective arrow.

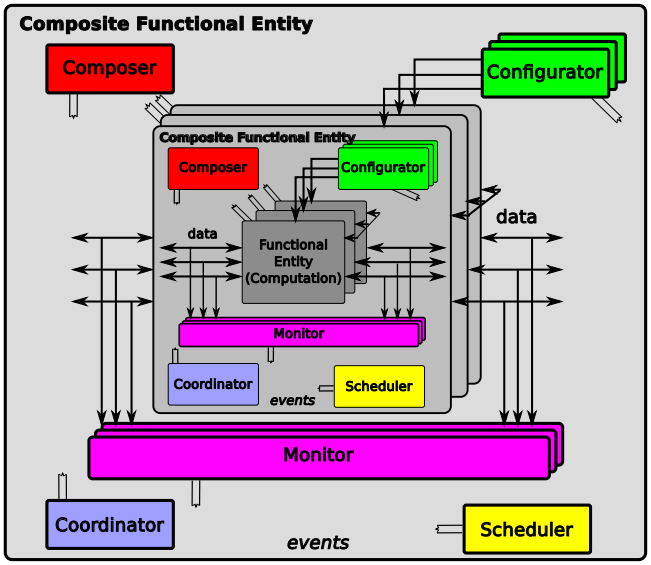

Fig. 2: Structure of the Composition Pattern. Blocks indicate entities, block colors indicate entity types, and darker shades of grey indicate deeper levels in the hierarchy. Arrows indicate data communication and double lines indicate event broadcasting over a 'bus system' (only partially drawn).

of the controller. The *Configurator* on its turn reacts to the event of the coordinator by setting a new control gain. The *Scheduler* of the reaching task composite first triggers the planner to generate a new setpoint or a complete trajectory, before it triggers the controller to track the setpoint or trajectory. The *Composer* models that the planner setpoint should be communicated to the controller. The *Communicator* models that the setpoint of the planner needs to be communicated in real-time (assuming online trajectory generation).

The presentation will detail this and other examples of the application of the Composition Pattern and its advantages over classical (e.g. layered) architectures to address the integration challenge in robotics. The Composition Pattern has

been applied to model robot applications that make use of constraint-based programming, for which a DSL is made available [3].

References

1. Bézivin, J.: On the unification power of models. Software and Systems Modeling 4(2), 171–188 (2005)
2. Bruyninckx, H., Klotzbücher, M., Hochgeschwender, N., Kraetzschmar, G., Gherardi, L., Brugali, D.: The BRICS Component Model: A model-based development paradigm for complex robotics software systems. In: 28th ACM Symposium On Applied Computing. pp. 1758–1764 (2013)
3. Vanthienen, D.: Composition Pattern for Constraint-based Programming with Application to Force-sensorless Robot Tasks. Ph.D. thesis, Dept. Mech. Eng., Katholieke Univ. Leuven, Belgium (January 2015)
4. Vanthienen, D., Klotzbücher, M., Bruyninckx, H.: The 5C-based architectural Composition Pattern: lessons learned from re-developing the iTaSC framework for constraint-based robot programming. J. Softw. Eng. in Robotics 5(1), 17–35 (2014)